1. Filter Structures
    1. Filter Structures
    2. FIR Filter Structures
    3. IIR Filter Structures
    4. State-Variable Representation of Discrete-Time Systems
2. Fixed-Point Numbers
    1. Fixed-Point Number Representation
    2. Fixed-Point Quantization
3. Quantization Error Analysis
    1. Finite-Precision Error Analysis
    2. Input Quantization Noise Analysis
    3. Quantization Error in FIR Filters
    4. Data Quantization in IIR Filters
    5. IIR Coefficient Quantization Analysis
4. Overflow Problems and Solutions
    1. Limit Cycles
    2. Scaling

## Filter Structures

A realizable filter must require only a finite number of computations per output sample. For linear, causal, time-Invariant filters, this restricts one to rational transfer functions of the form

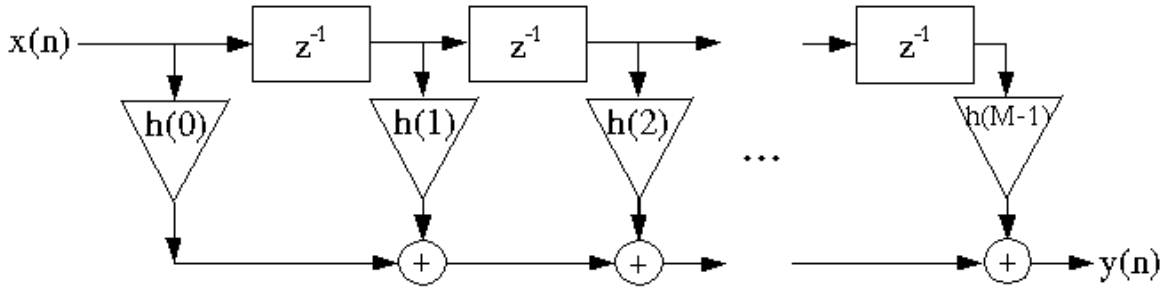
$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}}$$

Assuming no pole-zero cancellations, $H(z)$ is FIR if $\forall i, i > 0 : (a_i = 0)$, and IIR otherwise. Filter structures usually implement rational transfer functions as difference equations.

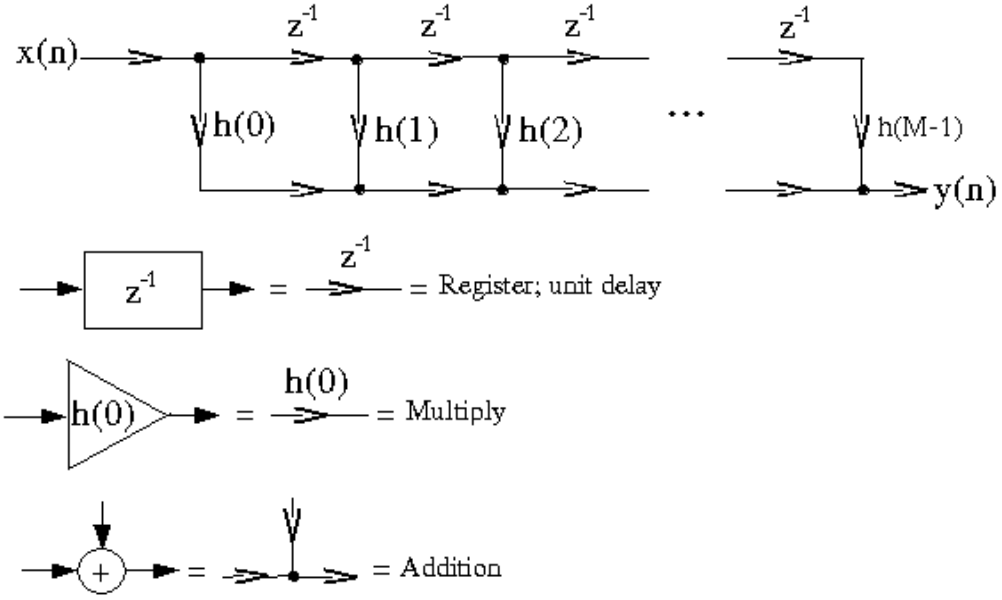
Whether FIR or IIR, a given transfer function can be implemented with many different filter structures. With infinite-precision data, coefficients, and arithmetic, all filter structures implementing the same transfer function produce the same output. However, different filter strucures may produce very different errors with quantized data and finite-precision or fixed-point arithmetic. The computational expense and memory usage may also differ greatly. Knowledge of different filter structures allows DSP engineers to trade off these factors to create the best implementation.

FIR Filter Structures

Consider causal FIR filters: $y(n) = \sum_{k=0}^{M-1} h(k)x(n-k)$; this can be realized using the following structure




or in a different notation




This is called the **direct-form FIR filter structure**.

There are no closed loops (no feedback) in this structure, so it is called a **non-recursive structure**. Since any FIR filter can be implemented using the direct-form, non-recursive structure, it is always possible to implement an FIR filter non-recursively. However, it is also possible to implement an FIR filter **recursively**, and for some special sets of FIR filter coefficients this is much more efficient.

**Example:**

$$y(n) = \sum_{k=0}^{M-1} x(n-k)$$

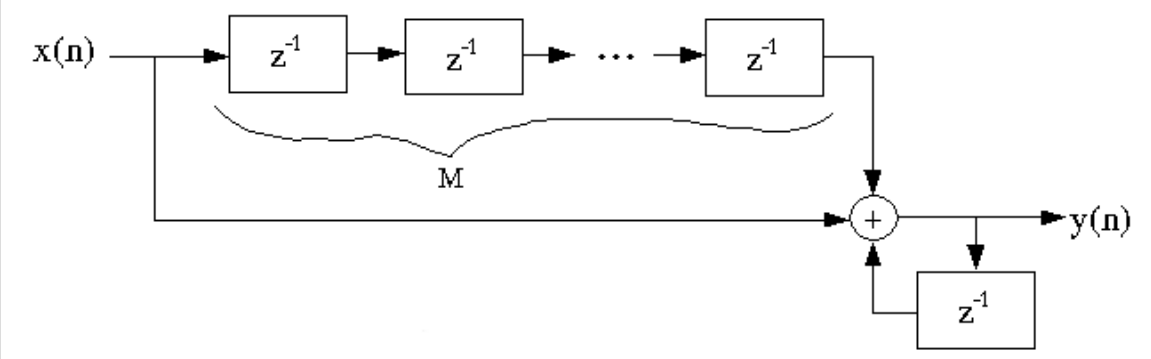
where

$$h(k) = \quad 0, 0, \underset{\hat{}}{1}_{k=0}, 1, \ldots, 1, \underset{\hat{}}{1}_{k=M-1}, 0, 0, 0, \ldots$$

But note that

$$y(n) = y(n-1) + x(n) - x(n-M)$$

This can be implemented as




Instead of costing $M-1$ adds/output point, this comb filter costs only two adds/output.

**Exercise:**

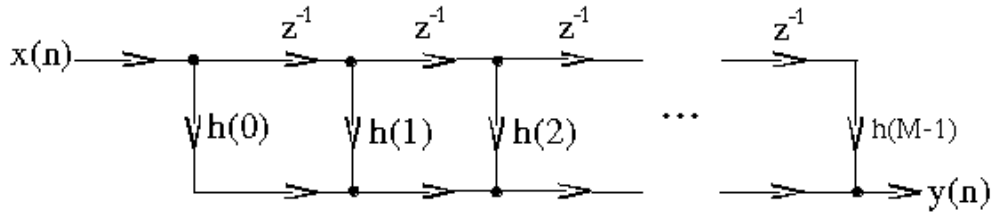
**Problem:** Is this stable, and if not, how can it be made so?

IIR filters must be implemented with a **recursive** structure, since that's the only way a finite number of elements can generate an infinite-length impulse response in a linear, time-invariant (LTI) system. Recursive structures have the advantages of being able to implement IIR systems, and sometimes greater computational efficiency, but the disadvantages of possible instability, limit cycles, and other deletorious effects that we will study shortly.

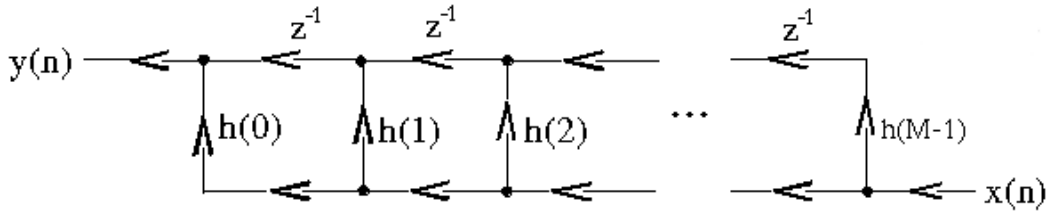
## Transpose-form FIR filter structures

The **flow-graph-reversal theorem** says that if one changes the directions of all the arrows, and inputs at the output and takes the output from the input of a reversed flow-graph, the new system has an identical input-output relationship to the original flow-graph.

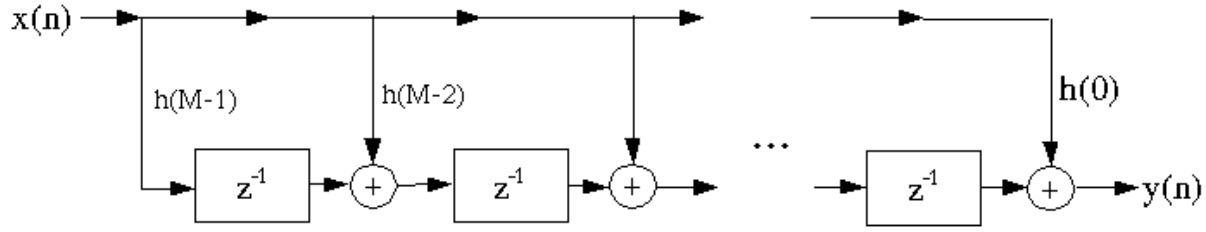
Direct-form FIR structure




reverse = transpose-form FIR filter structure




or redrawn




**Cascade structures**

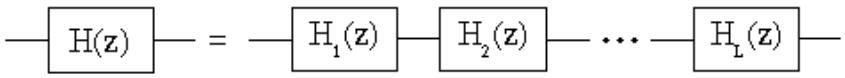
The z-transform of an FIR filter can be factored into a cascade of short-length filters

$$b_0 + b_1 z^{-1} + b_2 z^{-3} + \ldots + b_m z^{-m} = b_0 \left(1 - z_1 z^{-1}\right) \left(1 - z_2 z^{-1}\right) \ldots \left(1 - z_m z^{-1}\right)$$

where the $z_i$ are the zeros of this polynomial. Since the coefficients of the polynomial are usually real, the roots are usually complex-conjugate pairs, so we generally combine $\left(1 - z_i z^{-1}\right) \left(1 - z_i z^{-1}\right)$ into one quadratic (length-2) section with **real** coefficients

$$\left(1 - z_i z^{-1}\right) \left(1 - z_i z^{-1}\right) = 1 - 2\Re(z_i) z^{-1} + \left(|z_i|\right)^2 z^{-2} = H_i(z)$$

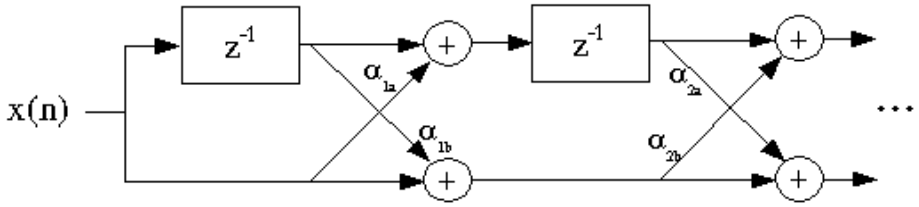
The overall filter can then be implemented in a **cascade** structure.




This is occasionally done in FIR filter implementation when one or more of the short-length filters can be implemented efficiently.

**Lattice Structure**

It is also possible to implement FIR filters in a lattice structure: this is sometimes used in adaptive filtering



IIR Filter Structures

IIR (Infinite Impulse Response) filter structures must be recursive (use feedback); an infinite number of coefficients could not otherwise be realized with a finite number of computations per sample.

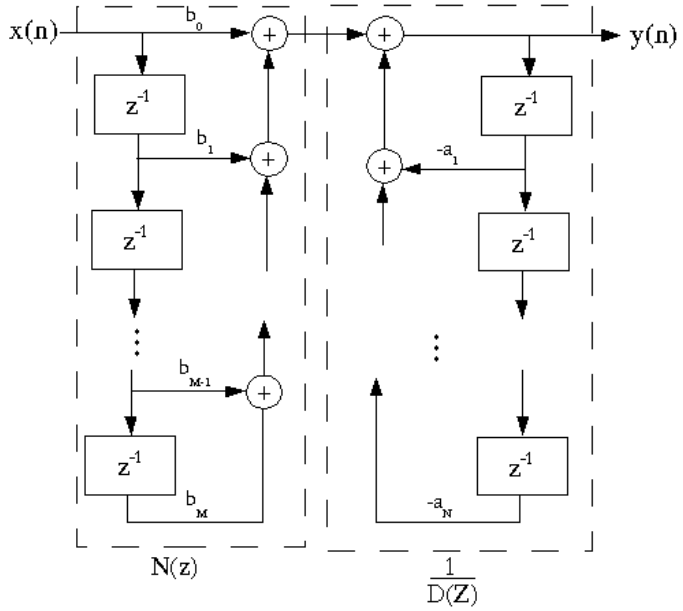
$$H(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

The corresponding time-domain difference equation is

$$y(n) = (-(a_1 y(n-1))) - a_2 y(n-2) + \ldots - a_N y(n-N) + b_0 x(0) + b_1 x(n-1) + \ldots + b_M x(n-M)$$

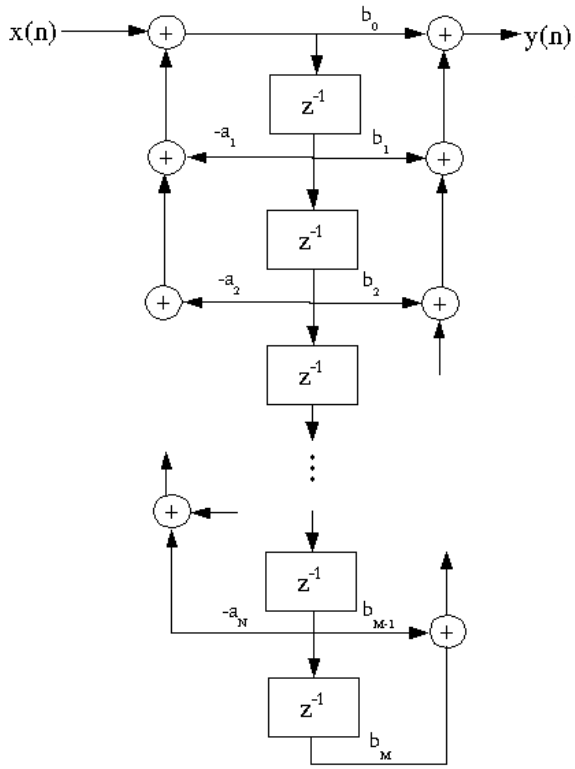
## Direct-form I IIR Filter Structure

The difference equation above is implemented directly as written by the Direct-Form I IIR Filter Structure.




Note that this is a cascade of two systems, $N(z)$ and $\frac{1}{D(z)}$. If we reverse the order of the filters, the overall system is unchanged: The memory elements appear in the middle and store identical values, so they can be combined, to form the Direct-Form II IIR Filter Structure.

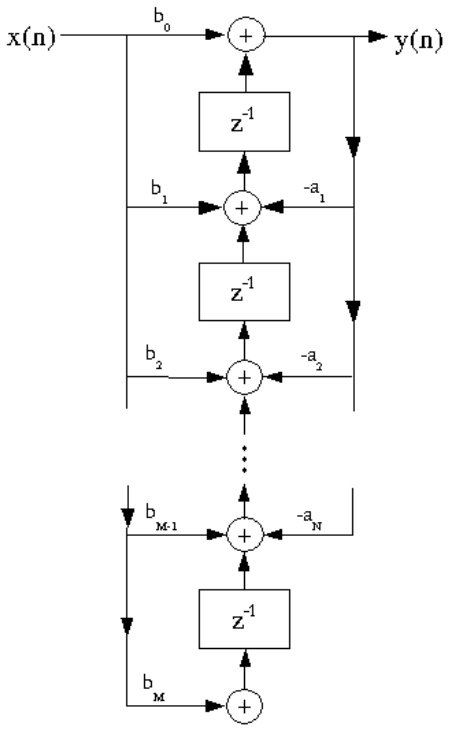
## Direct-Form II IIR Filter Structure




This structure is **canonic**: (i.e., it requires the minimum number of memory elements).

Flowgraph reversal gives the

### Transpose-Form IIR Filter Structure




Usually we design IIR filters with $N = M$, but not always.

Obviously, since all these structures have identical frequency response, filter structures are not unique. We consider many different structures because

1. Depending on the technology or application, one might be more convenient than another
2. The response in a practical realization, in which the data and coefficients must be **quantized**, may differ substantially, and some structures behave much better than others with quantization.

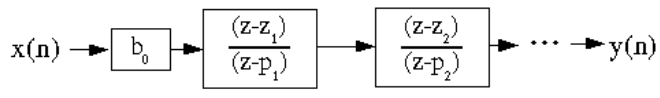
The Cascade-Form IIR filter structure is one of the least sensitive to quantization, which is why it is the most commonly used IIR filter structure.

## IIR Cascade Form

The numerator and denominator polynomials can be factored

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-m}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} = \frac{b_0 \prod_{k=1}^{M} z - z_k}{z^{M-N} \prod_{i=1}^{N} z - p_k}$$

and implemented as a cascade of short IIR filters.




Since the filter coefficients are usually real yet the roots are mostly complex, we actually implement these as second-order sections, where comple-conjugate pole and zero pairs are combined into second-order sections

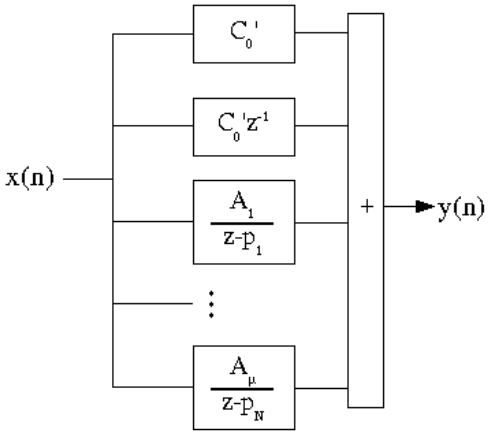
with real coefficients. The second-order sections are usually implemented with either the Direct-Form II or Transpose-Form structure.

## Parallel form

A rational transfer function can also be written as

$$\frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-m}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} = {c_0}' + {c_1}' z^{-1} + \ldots + \frac{A_1}{z - p_1} + \frac{A_2}{z - p_2} + \ldots + \frac{A_N}{z - p_N}$$

which by linearity can be implemented as




As before, we combine complex-conjugate pole pairs into second-order sections with real coefficients.

The cascade and parallel forms are of interest because they are much less sensitive to coefficient quantization than higher-order structures, as analyzed in later modules in this course.

## Other forms

There are many other structures for IIR filters, such as wave digital filter structures, lattice-ladder, all-pass-based forms, and so forth. These are the result of extensive research to find structures which are computationally efficient **and** insensitive to quantization error. They all represent various tradeoffs; the best choice in a given context is not yet fully understood, and may never be.

State-Variable Representation of Discrete-Time Systems

## State and the State-Variable Representation

State
: the minimum additional information at time $n$, which, along with all current and future input values, is necessary to compute all future outputs.

Essentially, the state of a system is the information held in the delay registers in a filter structure or signal flow graph.

**Note:** Any LTI (linear, time-invariant) system of finite order $M$ can be represented by a state-variable description

$$\boldsymbol{x}(n+1) = A\boldsymbol{x}(n) + \boldsymbol{B}u(n)$$

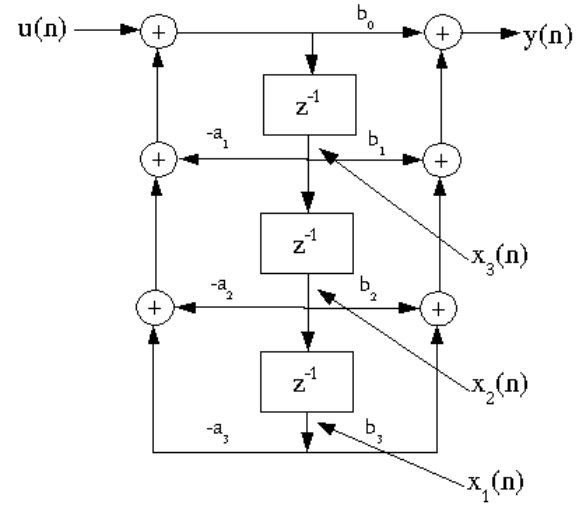
$$\boldsymbol{y}(n) = \boldsymbol{C}\boldsymbol{x}(n) + Du(n)$$

where $\boldsymbol{x}$ is an $M$ x 1 "state vector," $u(n)$ is the input at time $n$, $y(n)$ is the output at time $n$; $A$ is an $M$ x $M$ matrix, $\boldsymbol{B}$ is an $M$ x 1 vector, $\boldsymbol{C}$ is a 1 x $M$ vector, and $D$ is a 1 x 1 scalar.

One can always obtain a state-variable description of a signal flow graph.

**Example:**
**3rd-Order IIR**

$y(n) = (-(a_1 y(n-1))) - a_2 y(n-2) - a_3 y(n-3) + b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + b_3 x(n-3)$




$$\begin{matrix} x_1(n+1) \\ x_2(n+1) \\ x_3(n+1) \end{matrix} = \begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_3 & -a_2 & -a_1 \end{matrix} \begin{matrix} x_1(n) \\ x_2(n) \\ x_3(n) \end{matrix} + \begin{matrix} 0 \\ 0 \\ 1 \end{matrix} u(n)$$

$$y(n) = \begin{pmatrix} -(a_3 b_0) & -(a_2 b_0) & -(a_1 b_0) \end{pmatrix} \begin{matrix} x_1(n) \\ x_2(n) \\ x_3(n) \end{matrix} + (b_0)u(n)$$

**Exercise:**

**Problem:** Is the state-variable description of a filter $H(z)$ unique?

**Exercise:**

**Problem:** Does the state-variable description fully describe the signal flow graph?

## State-Variable Transformation

Suppose we wish to define a new set of state variables, related to the old set by a linear transformation: $\boldsymbol{q}(n) = T\boldsymbol{x}(n)$, where $T$ is a nonsingular $M$ x $M$ matrix, and $\boldsymbol{q}(n)$ is the new state vector. We wish the overall system to remain the same. Note that $\boldsymbol{x}(n) = T^{-1}\boldsymbol{q}(n)$, and thus

$$\boldsymbol{x}(n+1) = A\boldsymbol{x}(n) + Bu(n) \Rightarrow T^{-1}\boldsymbol{q}(n) = AT^{-1}\boldsymbol{q}(n) + Bu(n) \Rightarrow \boldsymbol{q}(n) = TAT^{-1}\boldsymbol{q}(n) + TBu(n)$$

$$y(n) = C\boldsymbol{x}(n) + Du(n) \Rightarrow y(n) = CT^{-1}\boldsymbol{q}(n) + Du(n)$$

This defines a new state system with an input-output behavior identical to the old system, but with different internal memory contents (states) and state matrices.

$$\boldsymbol{q}(n) = \hat{A}\boldsymbol{q}(n) + \hat{B}u(n)$$

$$y(n) = \hat{C}\boldsymbol{q}(n) + \hat{D}u(n)$$

$\hat{A} = TAT^{-1}$, $\hat{B} = TB$, $\hat{C} = CT^{-1}$, $\hat{D} = D$

These transformations can be used to generate a wide variety of alternative stuctures or implementations of a filter.

## Transfer Function and the State-Variable Description

Taking the $z$ transform of the state equations

$$Z[\boldsymbol{x}(n+1)] = Z[A\boldsymbol{x}(n) + Bu(n)]$$

$$Z[y(n)] = Z[C\boldsymbol{x}(n) + Du(n)]$$

$$\Downarrow$$

$$z\boldsymbol{X}(z) = A\boldsymbol{X}(z) + BU(z)$$

**Note:** $\boldsymbol{X}(z)$ is a vector of scalar $z$-transforms $\boldsymbol{X}(z)^T = \begin{pmatrix} X_1(z) & X_2(z) & \dots \end{pmatrix}$

$$\boldsymbol{Y}(z) = C\boldsymbol{X}(n) + DU(n)$$

$$(zI - A)\boldsymbol{X}(z) = BU(z) \Rightarrow \boldsymbol{X}(z) = (zI - A)^{-1}BU(z)$$

so
**Equation:**

$$\begin{aligned} Y(z) &= C(zI - A)^{-1}BU(z) + DU(z) \\ &= \Big(C(-(zI))^{-1}B + D\Big)U(z) \end{aligned}$$

and thus

$$H(z) = C(zI - A)^{-1}B + D$$

Note that since $(zI - A)^{-1} = \frac{\pm(\det{(zI-A)_{\text{red}}})^T}{\det(z(I)-A)}$, this transfer function is an $M$th-order rational fraction in $z$. The denominator polynomial is $D(z) = \det{(zI - A)}$. A discrete-time state system is thus stable if the $M$ roots of $\det{(zI - A)}$ (i.e., the poles of the digital filter) are all inside the unit circle.

Consider the transformed state system with $\hat{A} = TAT^{-1}$, $\hat{B} = TB$, $\hat{C} = CT^{-1}$, $\hat{D} = D$:
**Equation:**

$$\begin{aligned} H(z) &= \hat{C}\Big(zI - \hat{A}\Big)^{-1}\hat{B} + \hat{D} \\ &= CT^{-1}\big(zI - TAT^{-1}\big)^{-1}TB + D \\ &= CT^{-1}\big(T\,(zI - A)T^{-1}\big)^{-1}TB + D \\ &= CT^{-1}\big(T^{-1}\big)^{-1}(zI - A)^{-1}T^{-1}TB + D \\ &= C(zI - A)^{-1}B + D \end{aligned}$$

This proves that state-variable transformation doesn't change the transfer function of the underlying system. However, it can provide alternate forms that are less sensitive to coefficient quantization or easier to analyze, understand, or implement.

State-variable descriptions of systems are useful because they provide a fairly general tool for analyzing all systems; they provide a more detailed description of a signal flow graph than does the transfer function (although not a full description); and they suggest a large class of alternative implementations. They are even more useful in control theory, which is largely based on state descriptions of systems.

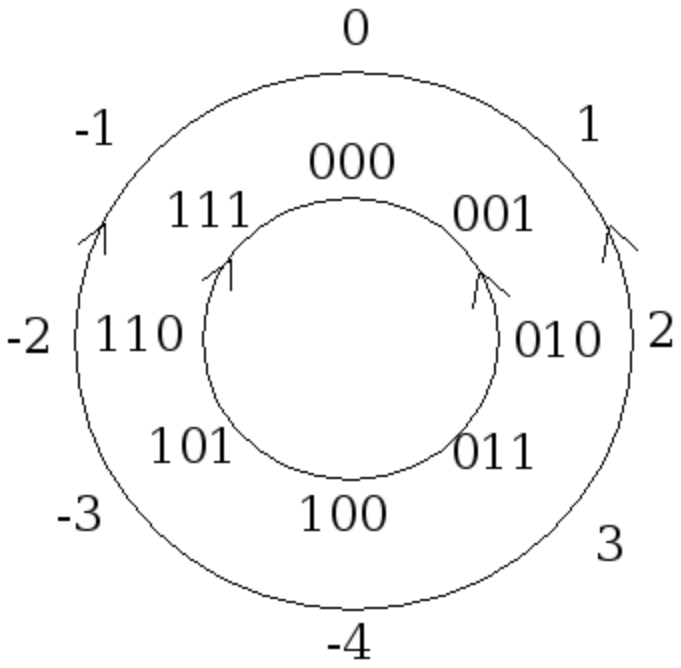
Fixed-Point Number Representation

Fixed-point arithmetic is generally used when hardware cost, speed, or complexity is important. Finite-precision quantization issues usually arise in fixed-point systems, so we concentrate on fixed-point quantization and error analysis in the remainder of this course. For basic signal processing computations such as digital filters and FFTs, the magnitude of the data, the internal states, and the output can usually be scaled to obtain good performance with a fixed-point implementation.

## Two's-Complement Integer Representation

As far as the hardware is concerned, fixed-point number systems represent data as $B$-bit integers. The two's-complement number system is usually used:

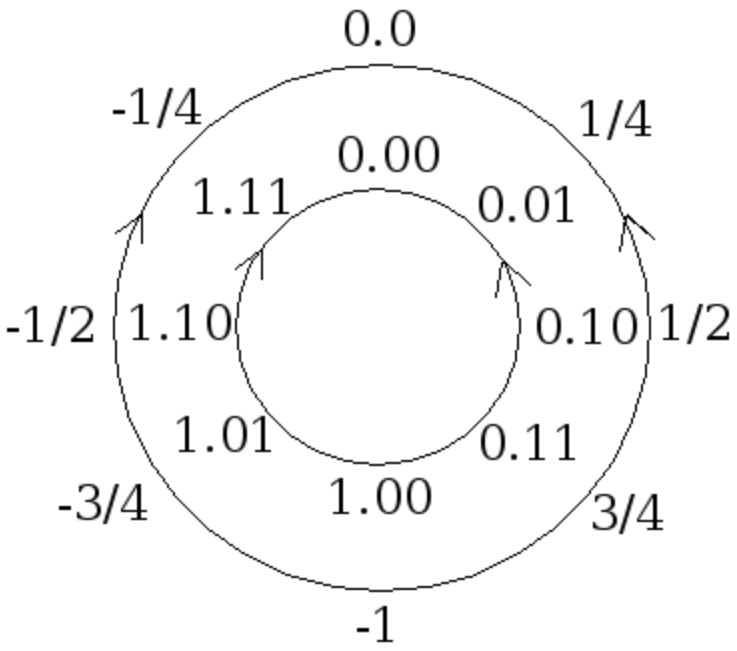
$$k = \begin{cases} \text{binary integer representation} \;\; \text{if} \;\; 0 \leq \text{k} \leq 2^{B-1} - 1 \\ \text{bit-by-bit inverse}(-k) + 1 \;\; \text{if} \;\; -2^{B-1} \leq k \leq 0 \end{cases}$$



The most significant bit is known at the **sign bit**; it is 0 when the number is non-negative; 1 when the number is negative.

## Fractional Fixed-Point Number Representation

For the purposes of signal processing, we often regard the fixed-point numbers as binary fractions between $[-1, 1)$, by implicitly placing a decimal point after the sign bit.



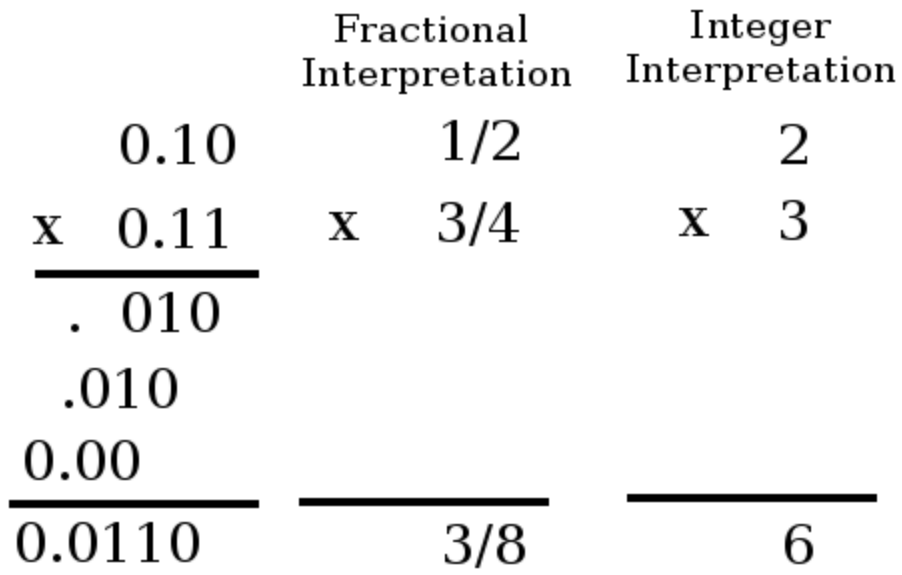

or

$$x = -b_0 + \sum_{i=1}^{B-1} b_i 2^{-i}$$

This interpretation makes it clearer how to implement digital filters in fixed-point, at least when the coefficients have a magnitude less than 1.

## Truncation Error

Consider the multiplication of two binary fractions




Note that full-precision multiplication almost doubles the number of bits; if we wish to return the product to a $B$-bit representation, we must truncate the $B-1$ least significant bits. However, this introduces **truncation error** (also known as **quantization error**, or **roundoff error** if the number is rounded to the nearest $B$-bit fractional value rather than truncated). Note that this occurs after **multiplication**.

## Overflow Error

Consider the addition of two binary fractions;

| | Fractional Interpretation | Integer Interpretation |
|---|---|---|
| 0.10 | 1/2 | 2 |
| + 0.11 | + 3/4 | + 3 |
| 1.01 | 5/4 = -1/4 | 5 = -1 |

Note the occurence of wraparound **overflow**; this only happens with **addition**. Obviously, it can be a bad problem.

There are thus two types of fixed-point error: roundoff error, associated with data quantization and multiplication, and overflow error, associated with data quantization and additions. In fixed-point systems, one must strike a balance between these two error sources; by scaling down the data, the occurence of overflow errors is reduced, but the relative size of the roundoff error is increased.

**Note:** Since multiplies require a number of additions, they are especially expensive in terms of hardware (with a complexity proportional to $B_x B_h$, where $B_x$ is the number of bits in the data, and $B_h$ is the number of bits in the filter coefficients). Designers try to minimize both $B_x$ and $B_h$, and often choose $B_x \neq B_h$!

Fixed-Point Quantization

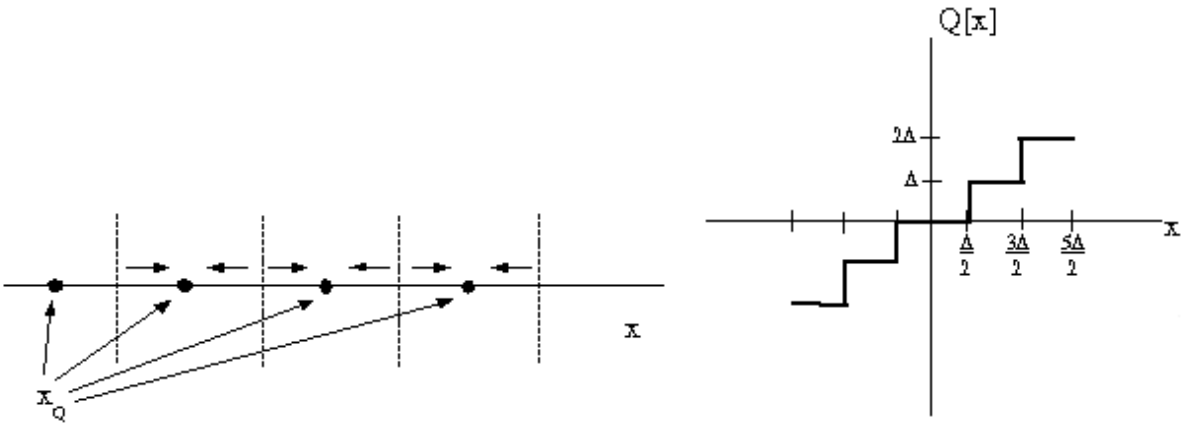
The fractional $B$-bit two's complement number representation evenly distributes $2^B$ quantization levels between $-1$ and $1 - 2^{-(B-1)}$. The spacing between quantization levels is then

$$\frac{2}{2^B} = 2^{-(B-1)} \doteq \Delta_B$$

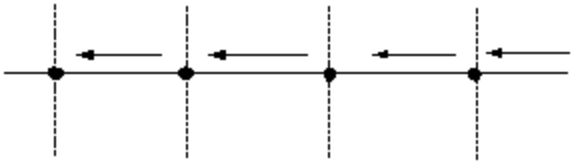
Any signal value falling between two levels is assigned to one of the two levels.

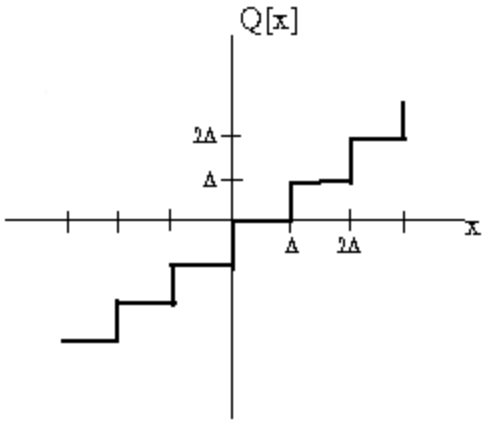
$X_Q = Q[x]$ is our notation for quantization. $e = Q[x] - x$ is then the quantization error.

One method of quantization is **rounding**, which assigns the signal value to the **nearest** level. The maximum error is thus $\frac{\Delta_B}{2} = 2^{-B}$.




Another common scheme, which is often easier to implement in hardware, is **truncation**. $Q[x]$ assigns $x$ to the next lowest level.

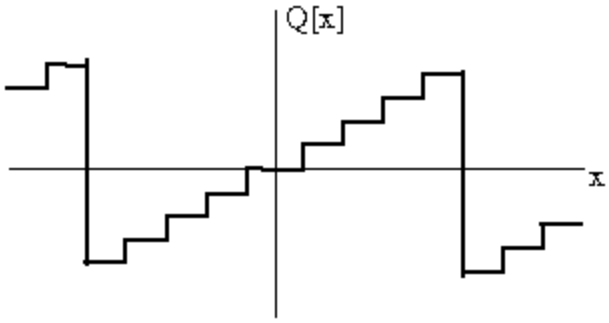




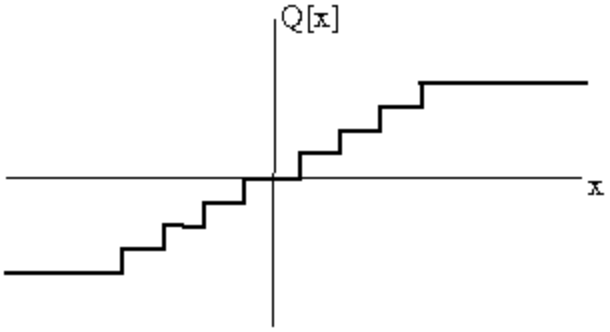

The worst-case error with truncation is $\Delta = 2^{-(B-1)}$, which is twice as large as with rounding. Also, the error is always negative, so on average it may have a non-zero mean (i.e., a bias component).

Overflow is the other problem. There are two common types: two's complement (or **wraparound**) overflow, or **saturation** overflow.

wraparound




saturation




Obviously, overflow errors are bad because they are typically **large**; two's complement (or wraparound) overflow introduces more error than

saturation, but is easier to implement in hardware. It also has the advantage that if the **sum** of several numbers is between $[-1, 1)$, the final answer will be correct even if intermediate sums overflow! However, wraparound overflow leaves IIR systems susceptible to zero-input large-scale limit cycles, as discussed in another module. As usual, there are many tradeoffs to evaluate, and no one right answer for all applications.

Finite-Precision Error Analysis

## Fundamental Assumptions in finite-precision error analysis

Quantization is a highly nonlinear process and is very difficult to analyze precisely. Approximations and assumptions are made to make analysis tractable.

### Assumption #1

The roundoff or truncation errors at any point in a system at each time are **random**, **stationary**, and **statistically independent** (white and independent of all other quantizers in a system).

That is, the error autocorrelation function is $r_e[k] = E[e_n e_{n+k}] = {\sigma_q}^2 \delta[k]$. Intuitively, and confirmed experimentally in some (but not all!) cases, one expects the quantization error to have a uniform distribution over the interval $\left[-\frac{\Delta}{2}, \frac{\Delta}{2}\right)$ for rounding, or $(-\Delta, 0]$ for truncation.

In this case, rounding has zero mean and variance

$$E[Q[x_n] - x_n] = 0$$

$${\sigma_Q}^2 = E\left[{e_n}^2\right] = \frac{{\Delta_B}^2}{12}$$

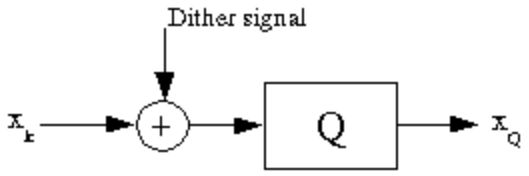
and truncation has the statistics

$$E[Q[x_n] - x_n] = -\frac{\Delta}{2}$$

$${\sigma_Q}^2 = \frac{{\Delta_B}^2}{12}$$

Please note that the independence assumption may be very bad (for example, when quantizing a sinusoid with an integer period $N$). There is

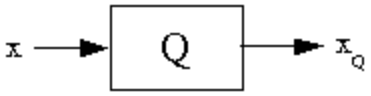
another quantizing scheme called **dithering**, in which the values are randomly assigned to nearby quantization levels. This can be (and often is) implemented by adding a small (one- or two-bit) random input to the signal before a truncation or rounding quantizer.



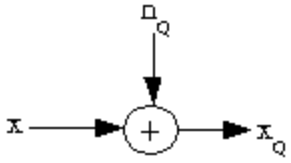

This is used extensively in practice. Altough the overall error is somewhat higher, it is spread evenly over all frequencies, rather than being concentrated in spectral lines. This is very important when quantizing sinusoidal or other periodic signals, for example.

**Assumption #2**

Pretend that the quantization error is really additive **Gaussian** noise with the same mean and variance as the uniform quantizer. That is, model




as



This model is a **linear** system, which our standard theory can handle easily. We model the noise as Gaussian because it remains Gaussian after passing through filters, so analysis in a system context is tractable.

## Summary of Useful Statistical Facts

- **correlation function** $r_x[k] \doteq E[x_n x_{n+k}]$
- **power spectral density** $S_x(w) \doteq \mathrm{DTFT}\,[r_x[n]]$
- Note $r_x[0] = {\sigma_x}^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} S_x(w)\ \mathrm{d}\ w$
- $r_{\mathrm{xy}}[k] \doteq E\left[x^*[n]y[n+k]\right]$
- **cross-spectral density** $S_{\mathrm{xy}}(w) = \mathrm{DTFT}\,[r_{\mathrm{xy}}[n]]$
- For $y = h^* x$:

$$S_{\mathrm{yx}}(w) = H(w)S_x(w)$$

$$S_{\mathrm{yy}}(w) = (|H(w)|)^2 S_x(w)$$

- Note that the **output** noise level after filtering a noise sequence is

$$ {\sigma_y}^2 = r_{\mathrm{yy}}[0] = \frac{1}{\pi}\int_{-\pi}^{\pi} (|H(w)|)^2 S_x(w)\ \mathrm{d}\ w$$

  so postfiltering quantization noise alters the noise power spectrum and may change its variance!
- For $x_1$, $x_2$ statistically independent

$$r_{x_1+x_2}[k] = r_{x_1}[k] + r_{x_2}[k]$$

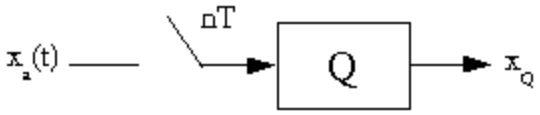
$$S_{x_1+x_2}(w) = S_{x_1}(w) + S_{x_2}(w)$$

- For independent random variables

$$ {\sigma_{x_1+x_2}}^2 = {\sigma_{x_1}}^2 + {\sigma_{x_2}}^2$$

Input Quantization Noise Analysis

All practical analog-to-digital converters (A/D) must quantize the input data. This can be modeled as an ideal sampler followed by a $B$-bit quantizer.




The signal-to-noise ratio (SNR) of an A/D is
**Equation:**

$$\begin{aligned} \mathrm{SNR} &= 10 \log \frac{P_x}{P_n} \\ &= 10 \log P_x - 10 \log \frac{{\Delta_B}^2}{12} \\ &= 10 \log P_x + 4.77 + 6.02B \end{aligned}$$

where $P_x$ is the power in the signal and $P_n$ is the power of the quantization noise, which equals its variance if it has a zero mean. The SNR increases by 6dB with each additional bit.

Quantization Error in FIR Filters

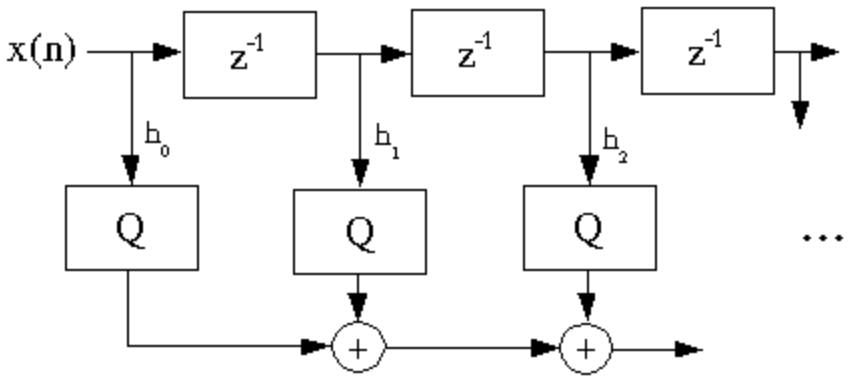
In digital filters, both the data at various places in the filter, which are continually varying, and the coefficients, which are fixed, must be quantized. The effects of quantization on data and coefficients are quite different, so they are analyzed separately.

## Data Quantization

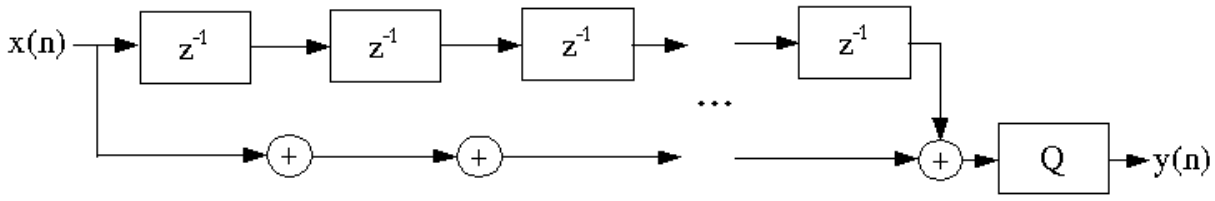
Typically, the input and output in a digital filter are quantized by the analog-to-digital and digital-to-analog converters, respectively. Quantization also occurs at various points in a filter structure, usually after a multiply, since multiplies increase the number of bits.

### Direct-form Structures

There are two common possibilities for quantization in a direct-form FIR filter structure: after each multiply, or only once at the end.




Single-precision accumulate; total variance $M\frac{\Delta^2}{12}$

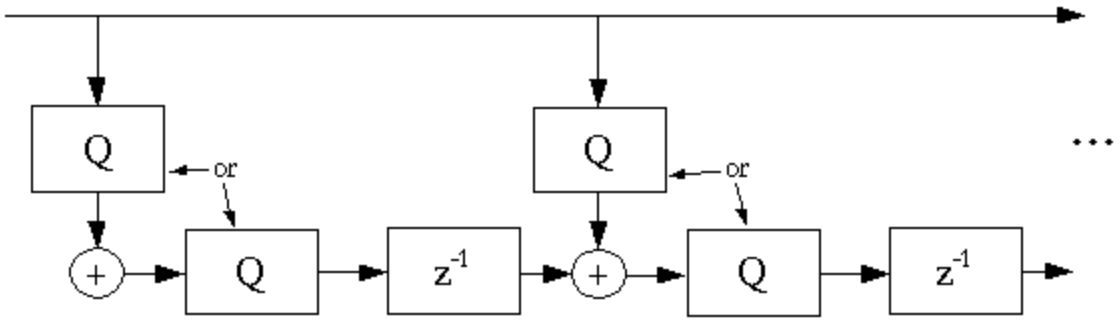



Double-precision accumulate; variance $\frac{\Delta^2}{12}$

In the latter structure, a double-length accumulator adds all $2B-1$ bits of each product into the accumulating sum, and truncates only at the end. Obviously, this is much preferred, and should **always** be used wherever possible. All DSP microprocessors and most general-pupose computers support double-precision accumulation.

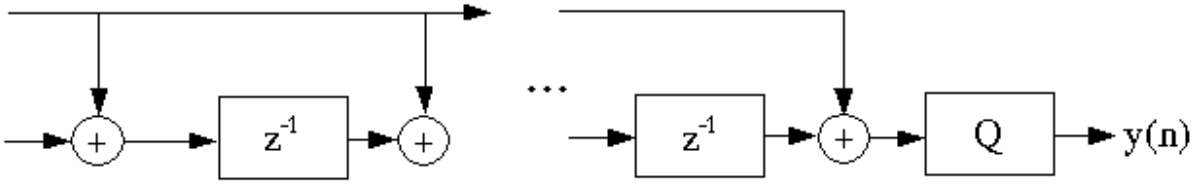
**Transpose-form**

Similarly, the transpose-form FIR filter structure presents two common options for quantization: after each multiply, or once at the end.




Quantize at each stage before storing intermediate sum. Output variance $M\frac{\Delta^2}{12}$

or




Store double-precision partial sums. Costs more memory, but variance $\frac{\Delta^2}{12}$

The transpose form is not as convenient in terms of supporting double-precision accumulation, which is a significant disadvantage of this structure.

## Coefficient Quantization

Since a quantized coefficient is fixed for all time, we treat it differently than data quantization. The fundamental question is: how much does the quantization affect the frequency response of the filter?

The quantized filter frequency response is

$$\mathrm{DTFT}\left[h_Q\right] = \mathrm{DTFT}\left[h_{\text{inf. prec.}} + e\right] = H_{\text{inf. prec.}}(w) + H_e(w)$$

Assuming the quantization model is correct, $H_e(w)$ should be fairly random and white, with the error spread fairly equally over all frequencies $w \in [-\pi, \pi)$; however, the randomness of this error destroys any equiripple property or any infinite-precision optimality of a filter.

**Exercise:**

**Problem:**

What quantization scheme minimizes the $L_2$ quantization error in frequency (minimizes $\int_{-\pi}^{\pi} \left(|H(w) - H_Q(w)|\right)^2 \, \mathrm{d}\, w$)? On average, how big is this error?

Ideally, if one knows the coefficients are to be quantized to $B$ bits, one should incorporate this directly into the filter design problem, and find the $M$ $B$-bit binary fractional coefficients minimizing the maximum deviation ( $L_\infty$ error). This can be done, but it is an integer program, which is known to be np-hard (i.e., requires almost a brute-force search). This is so expensive computationally that it's rarely done. There are some sub-optimal methods that are much more efficient and usually produce pretty good results.

Data Quantization in IIR Filters

Finite-precision effects are much more of a concern with IIR filters than with FIR filters, since the effects are more difficult to analyze and minimize, coefficient quantization errors can cause the filters to become unstable, and disastrous things like large-scale limit cycles can occur.

## Roundoff noise analysis in IIR filters

Suppose there are several quantization points in an IIR filter structure. By our simplifying assumptions about quantization error and Parseval's theorem, the quantization noise variance ${\sigma_{\mathrm{y,i}}}^2$ at the output of the filter from the $i$th quantizer is

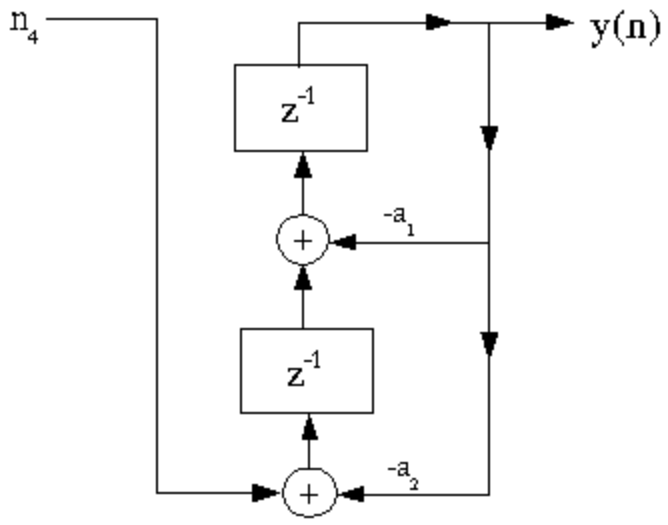
**Equation:**

$$\begin{aligned} {\sigma_{\mathrm{y,i}}}^2 &= \tfrac{1}{2\pi} \int_{-\pi}^{\pi} \left(|H_i(w)|\right)^2 SS_{n_i}(w) \, \mathrm{d}\, w \\ &= \tfrac{{\sigma_{n_i}}^2}{2\pi} \int_{-\pi}^{\pi} \left(|H_i(w)|\right)^2 \, \mathrm{d}\, w \\ &= {\sigma_{n_i}}^2 \sum_{n=-\infty}^{\infty} {h_i}^2(n) \end{aligned}$$

where ${\sigma_{n_i}}^2$ is the variance of the quantization error at the $i$th quantizer, $SS_{n_i}(w)$ is the power spectral density of that quantization error, and $HH_i(w)$ is the transfer function from the $i$th quantizer to the output point. Thus for $P$ independent quantizers in the structure, the total quantization noise variance is

$${\sigma_y}^2 = \frac{1}{2\pi} \sum_{i=1}^{P} {\sigma_{n_i}}^2 \int_{-\pi}^{\pi} \left(|H_i(w)|\right)^2 \, \mathrm{d}\, w$$

Note that in general, each $H_i(w)$, and thus the variance at the output due to each quantizer, is different; for example, the system as seen by a quantizer at the input to the first delay state in the Direct-Form II IIR filter structure to the output, call it $n_4$, is

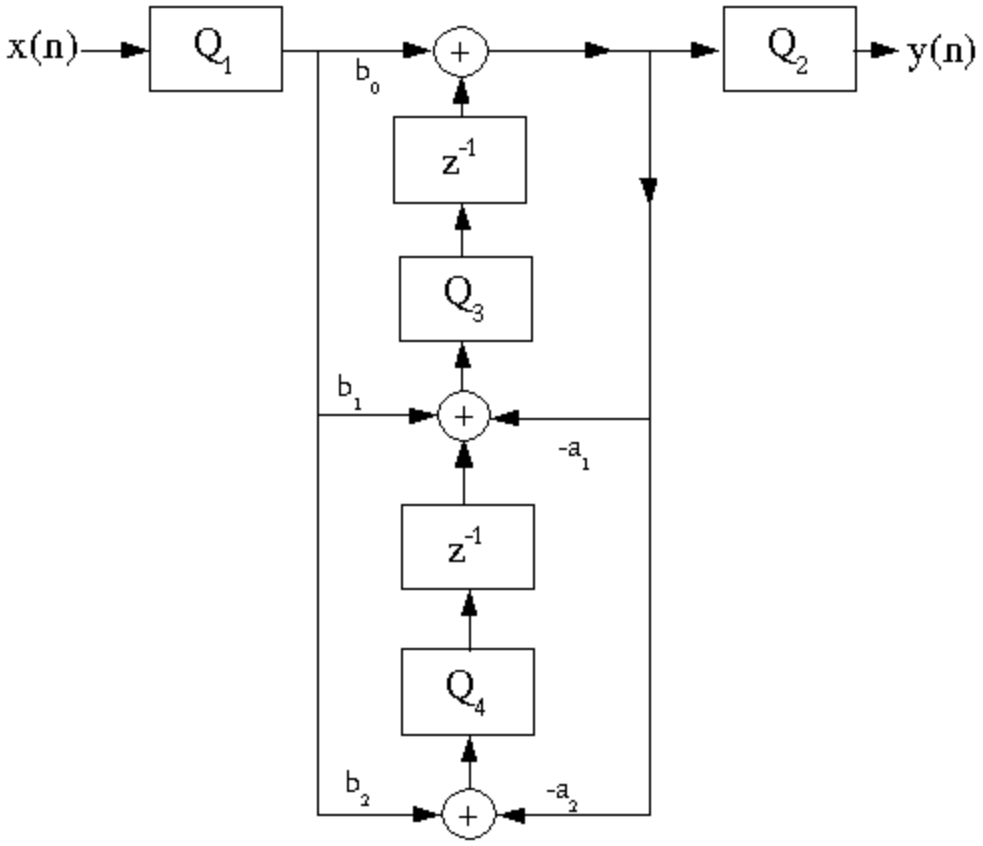



with a transfer function

$$H_4(z) = \frac{z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

which can be evaluated at $z = e^{iw}$ to obtain the frequency response.

A general approach to find $H_i(w)$ is to write state equations for the equivalent structure as seen by $n_i$, and to determine the transfer function according to $H(z) = C(zI - A)^{-1}B + d$.

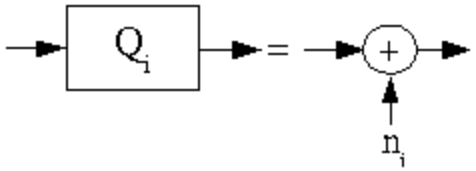



**Exercise:**

**Problem:**

The above figure illustrates the quantization points in a typical implementation of a Direct-Form II IIR second-order section. What is the total variance of the output error due to all of the quantizers in the system?

By making the assumption that each $Q_i$ represents a noise source that is white, independent of the other sources, and additive,



the variance at the output is the sum of the variances at the output due to each noise source:

$$\sigma_y{}^2 = \sum_{i=1}^{4} \sigma_{y,i}{}^2$$

The variance due to each noise source at the output can be determined from $\frac{1}{2\pi} \int_{-\pi}^{\pi} \left(|H_i(w)|\right)^2 S_{n_i}(w) \, \mathrm{d}\, w$; note that $S_{n_i}(w) = \sigma_{n_i}{}^2$ by our assumptions, and $H_i(w)$ is the transfer function **from the noise source to the output**.

IIR Coefficient Quantization Analysis

Coefficient quantization is an important concern with IIR filters, since straigthforward quantization often yields poor results, and because quantization can produce unstable filters.

## Sensitivity analysis

The performance and stability of an IIR filter depends on the pole locations, so it is important to know how quantization of the filter coefficients $a_k$ affects the pole locations $p_j$. The denominator polynomial is

$$D(z) = 1 + \sum_{k=1}^{N} a_k z^{-k} = \prod_{i=1}^{N} 1 - p_i z^{-1}$$

We wish to know $\frac{\partial p_i}{\partial a_k}$, which, for small deviations, will tell us that a $\delta$ change in $a_k$ yields an $\varepsilon = \delta \frac{\partial p_i}{\partial a_k}$ change in the pole location. $\frac{\partial p_i}{\partial a_k}$ is the **sensitivity** of the pole location to quantization of $a_k$. We can find $\frac{\partial p_i}{\partial a_k}$ using the chain rule.

$$\frac{\partial A(z)}{\partial a_k}\bigg|_{z=p_i} = \frac{\partial A(z)}{\partial z}\frac{\partial z}{\partial a_k}\bigg|_{z=p_i}$$

$$\Downarrow$$

$$\frac{\partial p_i}{\partial a_k} = \frac{\frac{\partial A(z_i)}{\partial a_k}\Big|_{z=p_i}}{\frac{\partial A(z_i)}{\partial z}\Big|_{z=p_i}}$$

which is
**Equation:**

$$\begin{aligned}\frac{\partial p_i}{\partial a_k} &= \frac{z^{-k}}{-\left(z^{-1}\prod_{j=j\neq i,1}^{N} 1-p_j z^{-1}\right)}\quad z=p_i \\ &= \frac{-{p_i}^{N-k}}{\prod_{j=j\neq i,1}^{N} p_j - p_i}\end{aligned}$$

Note that as the poles get closer together, the sensitivity increases greatly. So as the filter order increases and more poles get stuffed closer together inside the unit circle, the error introduced by coefficient quantization in the pole locations grows rapidly.

How can we reduce this high sensitivity to IIR filter coefficient quantization?

**Solution**

Cascade or parallel form implementations! The numerator and denominator polynomials can be factored off-line at very high precision and grouped into second-order sections, which are then quantized section by section. The sensitivity of the quantization is thus that of second-order, rather than $N$-th order, polynomials. This yields major improvements in the frequency response of the overall filter, and is almost always done in practice.

Note that the numerator polynomial faces the same sensitivity issues; the **cascade** form also improves the sensitivity of the zeros, because they are also factored into second-order terms. However, in the **parallel** form, the zeros are globally distributed across the sections, so they suffer from quantization of all the blocks. Thus the **cascade** form preserves zero locations much better than the parallel form, which typically means that the stopband behavior is better in the cascade form, so it is most often used in practice.

**Note:** On the basis of the preceding analysis, it would seem important to use cascade structures in FIR filter implementations. However, most FIR

filters are linear-phase and thus symmetric or anti-symmetric. As long as the quantization is implemented such that the filter coefficients retain symmetry, the filter retains linear phase. Furthermore, since all zeros off the unit circle must appear in groups of four for symmetric linear-phase filters, zero pairs can leave the unit circle only by joining with another pair. This requires relatively severe quantizations (enough to completely remove or change the sign of a ripple in the amplitude response). This "reluctance" of pole pairs to leave the unit circle tends to keep quantization from damaging the frequency response as much as might be expected, enough so that cascade structures are rarely used for FIR filters.

**Exercise:**

> **Problem:** What is the worst-case pole pair in an IIR digital filter?

---

> **Solution:**
>
> The pole pair closest to the real axis in the z-plane, since the complex-conjugate poles will be closest together and thus have the highest sensitivity to quantization.

## Quantized Pole Locations

In a direct-form or transpose-form implementation of a second-order section, the filter coefficients are quantized versions of the polynomial coefficients.

$$D(z) = z^2 + a_1 z + a_2 = (z - p)\,(z - p)$$

$$p = \frac{-a_1 \pm \sqrt{{a_1}^2 - 4a_2}}{2}$$

$$p = re^{i\theta}$$

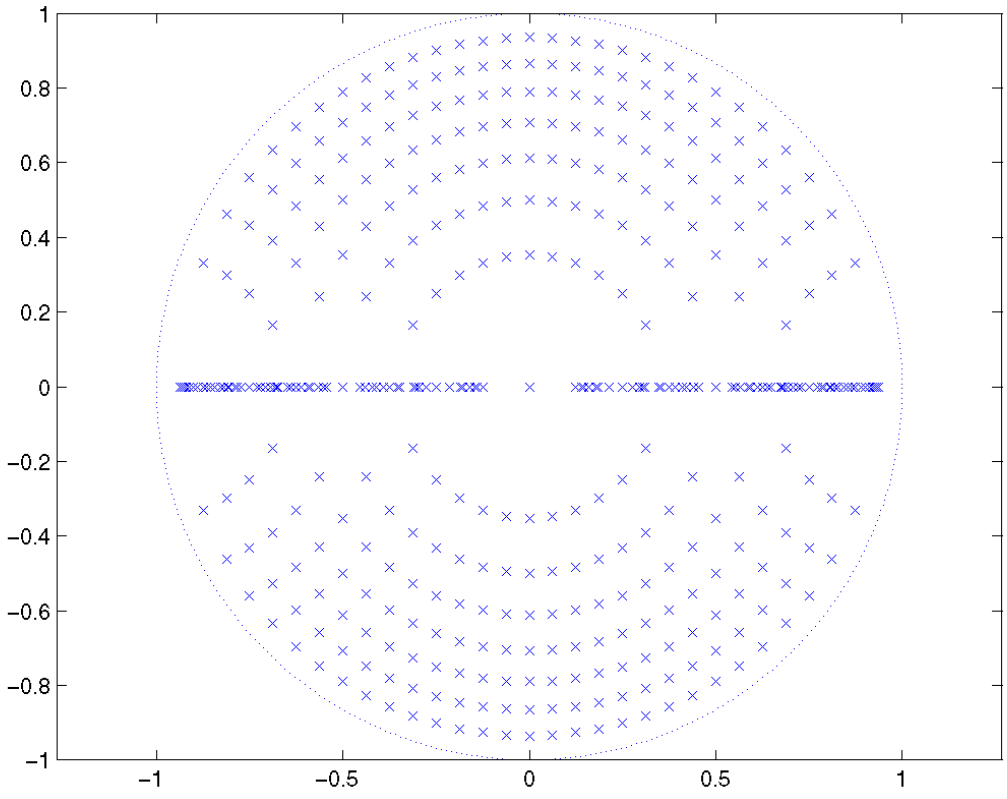
$$D(z) = z^2 - 2r\cos(\theta) + r^2$$

So

$$a_1 = -\left(2r\cos(\theta)\right)$$

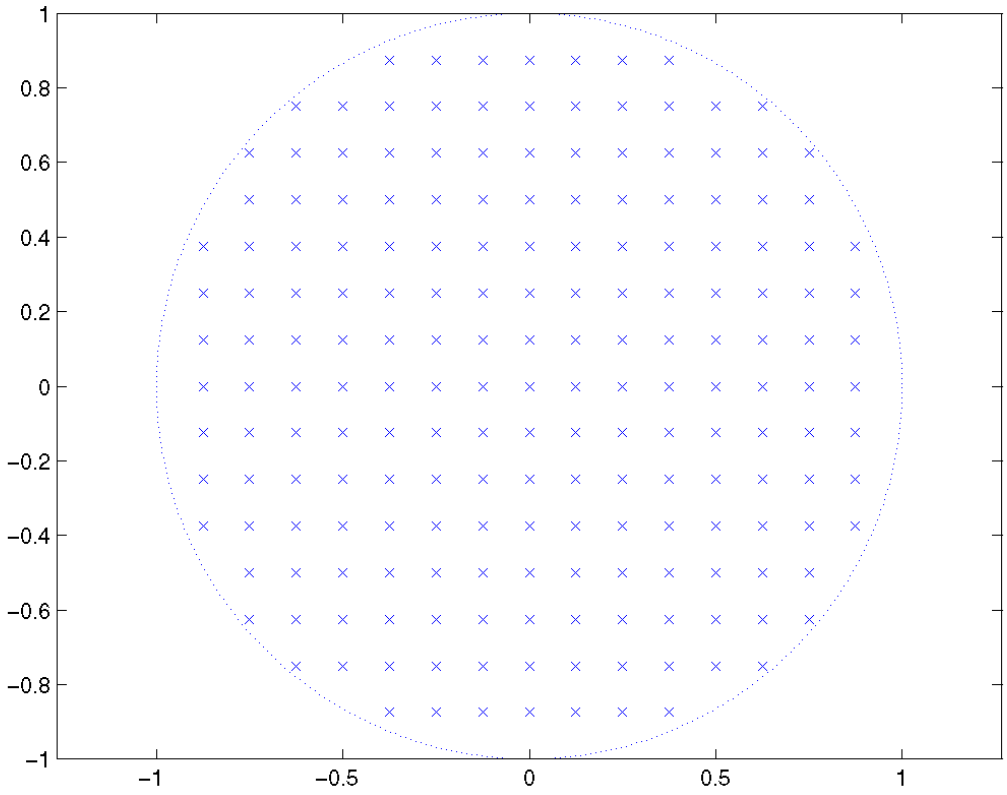
$$a_2 = r^2$$

Thus the quantization of $a_1$ and $a_2$ to $B$ bits restricts the radius $r$ to $r = \sqrt{k\Delta_B}$, and $a_1 = -\left(2\Re(p)\right) = k\Delta_B$ The following figure shows all stable pole locations after four-bit two's-complement quantization.




Note the nonuniform distribution of possible pole locations. This might be **good** for poles near $r = 1$, $\theta = \frac{\pi}{2}$, but not so good for poles near the origin or the Nyquist frequency.

In the "normal-form" structures, a state-variable based realization, the poles are uniformly spaced.




This can only be accomplished if the coefficients to be quantized equal the real and imaginary parts of the pole location; that is,

$$\alpha_1 = r\cos(\theta) = \Re(r)$$

$$\alpha_2 = r\sin(\theta) = \Im(p)$$

This is the case for a 2nd-order system with the state matrix $A = \begin{pmatrix} \alpha_1 & \alpha_2 \\ -\alpha_1 & \alpha_1 \end{pmatrix}$: The denominator polynomial is

**Equation:**

$$\begin{aligned}
\det\left(zI - A\right) &= \left(z - \alpha_1\right)^2 + {\alpha_2}^2 \\
&= z^2 - 2\alpha_1 z + {\alpha_1}^2 + {\alpha_2}^2 \\
&= z^2 - 2r\cos(\theta)z + r^2\left(\cos^2(\theta) + \sin^2(\theta)\right) \\
&= z^2 - 2r\cos(\theta)z + r^2
\end{aligned}$$

Given any second-order filter coefficient set, we can write it as a state-space system, find a transformation matrix $T$ such that $\hat{A} = T^{-1}AT$ is in normal form, and then implement the second-order section using a structure corresponding to the state equations.

The normal form has a number of other advantages; both eigenvalues are equal, so it minimizes the norm of $Ax$, which makes overflow less likely, and it minimizes the output variance due to quantization of the state values. It is sometimes used when minimization of finite-precision effects is critical.

**Exercise:**

> **Problem:** What is the disadvantage of the normal form?

---

> **Solution:**
>
> It requires more computation. The general state-variable equation requires nine multiplies, rather than the five used by the Direct-Form II or Transpose-Form structures.

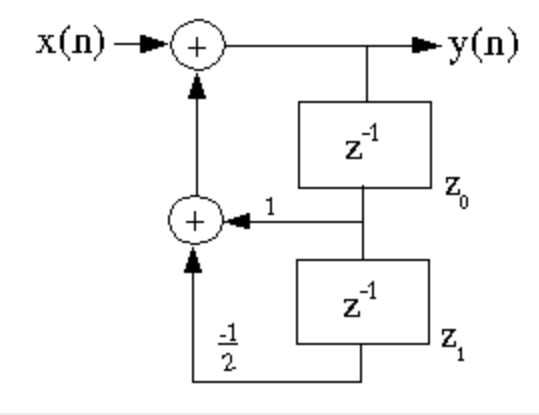
Limit Cycles

## Large-scale limit cycles

When overflow occurs, even otherwise stable filters may get stuck in a **large-scale limit cycle**, which is a short-period, almost full-scale persistent filter output caused by overflow.

**Example:**
Consider the second-order system

$$H(z) = \frac{1}{1 - z^{-1} + \frac{1}{2}z^{-2}}$$

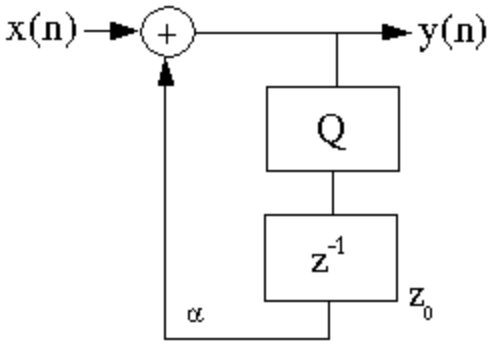



with zero input and initial state values $z_0[0] = 0.8$, $z_1[0] = -0.8$. Note $y[n] = z_0[n+1]$.
The filter is obviously stable, since the magnitude of the poles is $\frac{1}{\sqrt{2}} = 0.707$, which is well inside the unit circle. However, with wraparound overflow, note that $y[0] = z_0[1] = \frac{4}{5} - \frac{1}{2}\left(-\frac{4}{5}\right) = \frac{6}{5} = -\frac{4}{5}$, and that $z_0[2] = y[1] = \left(-\frac{4}{5}\right) - \frac{1}{2}\frac{4}{5} = -\frac{6}{5} = \frac{4}{5}$, so $y[n] = -\frac{4}{5}, \frac{4}{5}, -\frac{4}{5}, \frac{4}{5}, \ldots$ even with zero input.

Clearly, such behavior is intolerable and must be prevented. Saturation arithmetic has been proved to prevent **zero-input limit cycles**, which is one reason why all DSP microprocessors support this feature. In many applications, this is considered sufficient protection. Scaling to prevent overflow is another solution, if as well the inital state values are never initialized to limit-cycle-producing values. The normal-form structure also reduces the chance of overflow.

## Small-scale limit cycles

**Small-scale limit cycles** are caused by quantization. Consider the system




Note that when $\alpha z_0 > z_0 - \frac{\Delta_B}{2}$, rounding will quantize the output to the current level (with zero input), so the output will remain at this level forever. Note that the maximum amplitude of this "small-scale limit cycle" is achieved when

$$\alpha z_0 = z_0 - \frac{\Delta_B}{2} \Longrightarrow = z_{\max} = \frac{\Delta_B}{2 \times (1 - \alpha)}$$

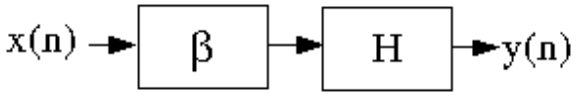
In a higher-order system, the small-scale limit cycles are oscillatory in nature. Any quantization scheme that never increases the magnitude of any quantized value prevents small-scale limit cycles.

**Note:** Two's-complement truncation does **not** do this; it increases the magnitude of negative numbers.

However, this introduces greater error and bias. Since the level of the limit cycles is proportional to $\Delta_B$, they can be reduced by increasing the number of bits. Poles close to the unit circle increase the magnitude and likelihood of small-scale limit cycles.

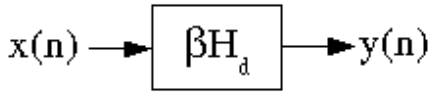
Scaling

Overflow is clearly a serious problem, since the errors it introduces are very large. As we shall see, it is also responsible for large-scale limit cycles, which cannot be tolerated. One way to prevent overflow, or to render it acceptably unlikely, is to **scale** the input to a filter such that overflow cannot (or is sufficiently unlikely to) occur.




In a fixed-point system, the range of the input signal is limited by the fractional fixed-point number representation to $|x[n]| \leq 1$. If we scale the input by multiplying it by a value $\beta$, $0 < \beta < 1$, then $|\beta x[n]| \leq \beta$.

Another option is to incorporate the scaling directly into the filter coefficients.




## FIR Filter Scaling

What value of $\beta$ is required so that the output of an FIR filter cannot overflow ($\forall n : (|y(n)| \leq 1)$, $\forall n : (|x(n)| \leq 1)$)?

$$|y(n)| = \left| \sum_{k=0}^{M-1} h(k)\beta x(n-k) \right| \leq \sum_{k=0}^{M-1} |h(k)|\,|\beta|\,|x(n-k)| \leq \beta \sum_{k=0}^{M-1} |h(k)|1$$

$$\Downarrow$$

$$\beta < \sum_{k=0}^{M-1} |h(k)|$$

Alternatively, we can incorporate the scaling directly into the filter, and require that

$$\sum_{k=0}^{M-1} |h(k)| < 1$$

to prevent overflow.

## IIR Filter Scaling

To prevent the output from overflowing in an IIR filter, the condition above still holds: ($M = \infty$)

$$|y(n)| < \sum_{k=0}^{\infty} |h(k)|$$

so an initial scaling factor $\beta < \frac{1}{\sum_{k=0}^{\infty}|h(k)|}$ can be used, or the filter itself can be scaled.

However, it is also necessary to prevent the **states** from overflowing, and to prevent overflow at any point in the signal flow graph where the arithmetic hardware would thereby produce errors. To prevent the states from overflowing, we determine the transfer function from the input to all states $i$, and scale the filter such that $\forall i : (\sum_{k=0}^{\infty} |h_i(k)| \leq 1)$

Although this method of scaling guarantees no overflows, it is often too conservative. Note that a worst-case signal is $x(n) = \text{sign}\,(h(-n))$; this input may be extremely unlikely. In the relatively common situation in which the input is expected to be mainly a single-frequency sinusoid of unknown frequency and amplitude less than 1, a scaling condition of

$$\forall w : (|H(w)| \leq 1)$$

is sufficient to guarantee no overflow. This scaling condition is often used. If there are several potential overflow locations $i$ in the digital filter structure, the scaling conditions are

$$\forall i, w : (|H_i(w)| \leq 1)$$

where $H_i(w)$ is the frequency response from the input to location $i$ in the filter.

Even this condition may be excessively conservative, for example if the input is more-or-less random, or if occasional overflow can be tolerated. In practice, experimentation and simulation are often the best ways to optimize the scaling factors in a given application.

For filters implemented in the cascade form, rather than scaling for the entire filter at the beginning, (which introduces lots of quantization of the input) the filter is usually scaled so that each stage is just prevented from overflowing. This is best in terms of reducing the quantization noise. The scaling factors are incorporated either into the previous or the next stage, whichever is most convenient.

Some heurisitc rules for grouping poles and zeros in a cascade implementation are:

1. Order the poles in terms of decreasing radius. Take the pole pair closest to the unit circle and group it with the zero pair closest to that pole pair (to minimize the gain in that section). Keep doing this with all remaining poles and zeros.
2. Order the section with those with highest gain ($\operatorname{argmax} |H_i(w)|$) in the middle, and those with lower gain on the ends.

Leland B. Jackson has an excellent intuitive discussion of finite-precision problems in digital filters. The book by Roberts and Mullis is one of the most thorough in terms of detail.